

PICXEX18 AND PIC

Lourens le Grange

bitcraft@global.co.za

This article describes the development of a multi-tasking operating system for the PIC18xxx micro controller from Microchip. The code and RAM memory requirements are minimal and the basic idea straightforward, but it can simplify the implementation of an application on this micro controller considerably.

Applications are easier to specify, design, code and debug if you start off by separating it into functional elements, or call it tasks. A software structure to handle the tasks could then look like **Figure 1**.

Here our 'operating system' is nothing more than a couple of CALL instructions with a GOTO MAIN instruction at the end. The CALL instruction activates the task, and the task must cooperate by executing a RETURN instruction to pass control back to the operating system. The above could do for some small applications, but usually the functions/tasks making up our application require some form of control over their own execution and that of other tasks. For example a key-scanner task could continuously monitor a few push buttons. When one is operated the task associated with that button needs to be activated by the key-scanner task. Or a sequence (task) could be executing one of its (many)

steps and then proceed to the next step (code section) once a particular condition is met.

It is clear that we will require more than RETURN instructions for the scanner and sequence tasks above to cooperate in a system. The key-scanner task needs to activate one of a few dormant (switched OFF) tasks, while the sequence task must have the ability to modify its own entry point so that only a particular section of its code (i.e. the active step and its criteria) is executed at a time. A task's entry point is the memory location at which the task's code will start executing when the operating system passes control to that task.

PICXEX18 functionality

In PICXEX18 the following functions are provided in the form of macros that the programmer uses in his task code.

XexTaskON TaskNr

With this macro the user can activate any task in the application. It can also be used in code outside a task, for instance an interrupt service routine can start a task.

XexTaskOFF TaskNr

With this macro the user can deactivate any task in the application. It can also be used in any code.

XexTaskExit

A macro used in a task to pass control back to the operating system.

XexSetEPPoint TaskNr, EPPoint

Modify a task's entry point. It can also be used in any code.

XexEPNext TaskNr

Set a task's entry point to the next instruction. Only to be

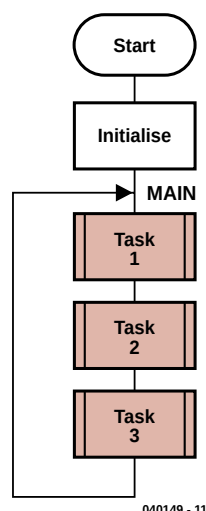


Figure 1.
A basic software
structure for
implementing an
application that is
divided into 3
functional elements,
or tasks.

040149 - 11

used in the code of the task of which the entry point is to be modified.

In Figure 1 the operating system's 'task activation code' is a single CALL TASK_X instruction. This will have to be extended. Firstly we need to add a test to prevent a task from being activated if it is switched OFF. Secondly we must be able to change a task's entry point during run-time, in which case a CALL instruction cannot be used as the destination address of this instruction is fixed at assembly time.

The ON/OFF test for a task is implemented by assigning a bit variable to each CYCLE TASK, which if set, will indicate that the task is in the active state. The facility to modify a task's entry point is implemented by using the PIC18's ability to push an address on its return stack. So the CYCLE TASK activation code needs to first test the task's ON/OFF flag, if it is set then get the task's current entry point value, push it on the Return Stack, and execute a RETURN instruction.

For each CYCLE TASK 3 RAM bytes are reserved. The first byte holds the ON/OFF flag in its Bit<7>, and the next two bytes the task's current Entry Point value.

Application design can be further simplified by adding to our operating system the ability to execute tasks at selectable time rates. This allows the programmer to concentrate on the task code itself without having to be concerned about how to code timing or counting mechanisms to regulate execution of that code.

PICEX18 allows for 8 TIME TASKS with the execution rate of each task being set at assembly time in the file xextask1.def. As their execution depends on time these tasks cannot be switched ON or OFF by the user, and their entry points cannot be modified. PICEX18 uses a CALL instruction to activate a TIME TASK. This allows for a simple activation mechanism — if the task's ON/OFF flag is set then CALL the task's code. To pass control back to the operating system the programmer uses the Xex-TaskExit macro inside the task code. If you look at the actual code for this macro you will see that a RETURN instruction will do the same job. Using the macro is just a better way of indicating in your code where the task's exit point is.

The basics of the PICEX18 multitasker can now be described by referring to **Figure 2**. Before a CYCLE TASK is activated a test is made to determine if there are any TIME TASKS waiting to be executed. If there is, then control is passed (with a CALL MAIN01 instruction) to the code in **Figure 3**.

Here those TIME TASKS, of which the Task Request Bits are set, will be executed. At the end of the TIME TASK list is a RETURN instruction, which will take us back into the CYCLE TASK loop of Figure 2, so execution of CYCLE TASKS carry on from where it left off.

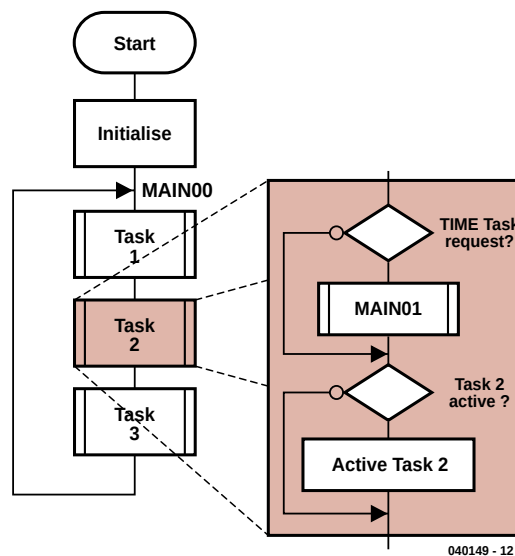


Figure 2.
Main Cycle Task
loop of PICEX18
showing how the
test for TIME TASK
request is done
before a CYCLE
TASK is activated.

As shown in Figure 2 we test for requests to execute TIME TASKS between the executions of CYCLE TASKS. It is therefore important for CYCLE TASKS not to retain control of the CPU for longer than about 5 ms. A PIC running at a modest 8 MHz can get through a lot of code in 5 ms! A scheduler subroutine in PICEX18 controls the activation of TIME TASKS by setting the Task Request Bits at the required times.

For the scheduler to function your code must provide for a timer routine to CALL the scheduler subroutine every 10 ms.

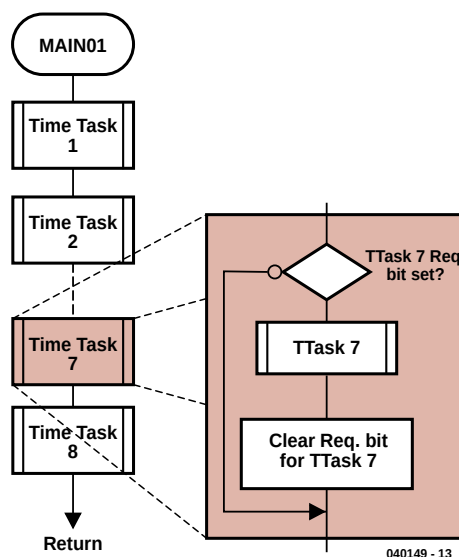


Figure 3. The
operating system
will execute a CALL
MAIN01 instruction
if any TIME TASK
Request Bit is set.
MAIN01 will
activate all TIME
TASKS of which the
Request Bit is set.

Figure 4.
For the TIME TASK scheduler of PICXEX18 to function the user must arrange for a timer routine to execute a CALL SCHED00 instruction every 10 ms.

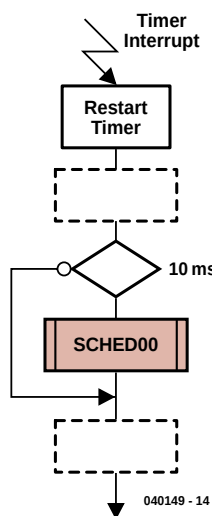


Figure 4 gives the basics of a timer interrupt service routine (which is your code) containing a call to the subroutine SCHED00. The following execution rates (in ms) are available for Time Tasks:

10, 20, 40, 50, 100, 200

The time task scheduler uses a 20-entry look-up table to save the task timing information. Every 10 ms the scheduler routine fetches the next entry in this table and OR it to the RAM location holding the 8 TIME TASK request bits. The entries in the look up table are generated at assembly time using information from the file xextask1.def. The source code for PICXEX18 is contained in the following three files:

Xextask1.def
Xex18tsk.mac
Xex18mak.asm

Xextask1.def

This is the only file that requires modification. The file is used to define the following:
Number of CYCLE TASKS.
Label at the start of each TIME TASK.
Execution rate for each TIME TASK.
Bias time for each TIME TASK.

Xex18tsk.mac

Source code for task control macros.

Xex18mak.asm

This file contains the code for PICXEX18. When this file is assembled it will generate the CYCLE and TIME TASK loops and the scheduler table.

Using PICXEX18

The archive file **040149-11.zip** you can download for free from our website contains all the source code required for the following example application to demonstrate the use of PICXEX18. To generate the .HEX file simply assemble the file ELEKTOR.ASM using the MPASM assembler (free from Microchip). The latter file contains

Figure 5.
Functional diagram of the example application, a combined Pulse/Ramp Generator.

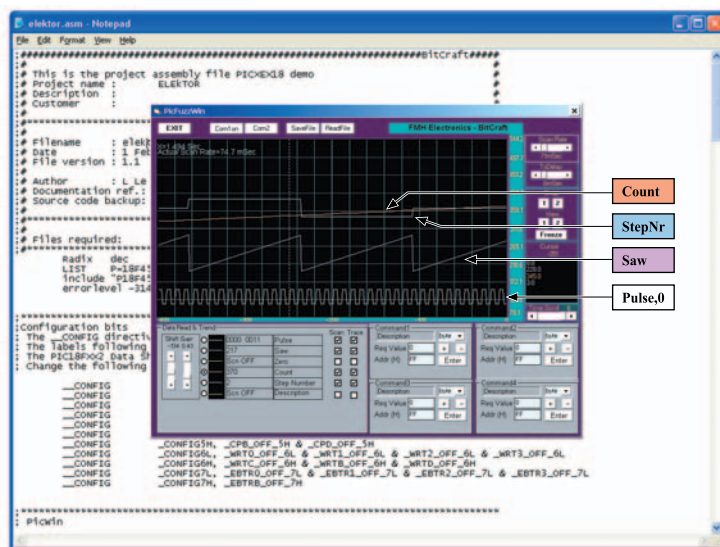
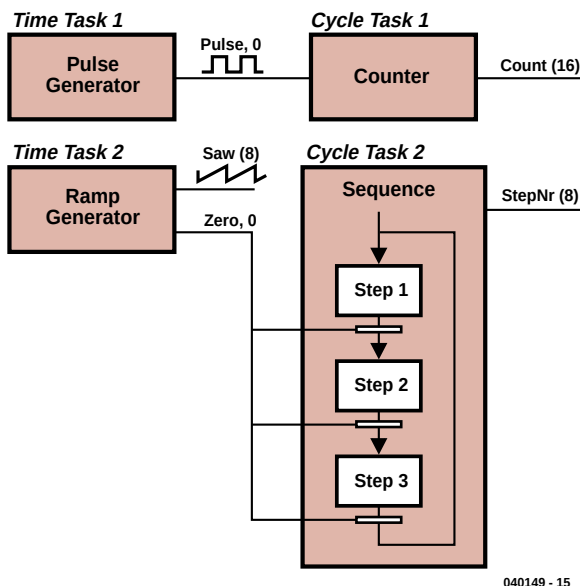


Figure 6.
PicWin8 monitor in action. Here, the 'Fuzzy' version is shown (still under development).

Free Downloads

All assembler files for PICXEX18 and the Pulse/Ramp demo program.
PICWin8 executable and user guide.

File number: 040149-11.zip

www.elektor-electronics.co.uk/dl/dl.htm,
select month of publication.

the directives to include the other files required. The application contains two Cycle Tasks and two Time Tasks.

Time Task 1 is a pulse generator. It will pulse the bit 'Pulse,0' at a rate of one pulse every 400 ms.

Time Task 2 is a sawtooth generator. It will cause the byte 'Saw' to ramp up in 2550 ms to a value of 255 and then reset to 0. On every reset the bit 'Zero,0' is set (it will be reset by the active step of the sequence).

Cycle Task 1 is a pulse counter. Every rising edge on the bit 'Pulse,0' will increment the 16-bit value 'Count'.

Cycle Task 2 is a sequence with three steps. The sequence will remain in a step until the bit 'Zero,0' is set. This bit will be cleared and the sequence advance to the next step. When in the third step and bit 'Zero,0' is set, then the sequence will revert back to step number 1. The byte 'StepNr' indicates the number of the current step. Once you have assembled this application, it can be tested by running it using the MPLAB simulator. To verify that the two time tasks are in fact being executed at the required rates of 200 and 40 ms, use a breakpoint at the tasks's starting point (PULSE_GEN and RAMP_GEN) and MPLAB's Stopwatch function to see how long it takes between executions. The application was programmed into a PIC18F452 clocked at 8 MHz.

PICWin8

When testing an application where a few things are happening at the same time it is very helpful to be able to

see how the different signals are behaving. In the ELEKTOR.ASM file, right after the setting of the configuration bits, you'll find a statement to set the assembler variable 'PicWin'. This is used to include the assembly code for an interrupt service routine for the UART to handle the data exchange with a PC utility to monitor and modify variables in the PIC. The screenshot of the PicWin monitor in action, **Figure 6**, shows trend traces of the four variables of our application, which, as you can see, is functioning as required.

The 'PicWin' monitor has the following features:

- VB6 application
- Baud rate 9600 (fixed)
- Can display & trend up to 6 values simultaneously
- 4 fields supplied for commands (modify PIC RAM locations).
- Each trend buffer = 800 samples. Display = 400 samples with scroll
- Data types: bit, byte, word, Fpoint
- Adjustable scan rate
- Performance: the screenshot shows four traces scanned at 87 ms. Each trace was shifted to get a better view of what's happening.

A user guide and setup instructions for the PicWin monitor may be found in a text file included in the zip archive for this project. Happy Pic'king!

(040149-1)

Advertisement